



UNDERSTANDING UI

- ▶ Web application UI is ultimately HTML/CSS/JavaScript
- ▶ Let R users write user interfaces using a simple, familiar-looking API...
- ▶ ...but no limits for advanced users

Ladder of progression

LADDER OF UI PROGRESSION

- Step 1. Shiny built-in inputs/outputs and layouts (sidebarLayout, navbarPage, tabsetPanel)
- Step 2. Use functions from external packages (shinythemes, shinydashboard, shinybs)
- Step 3. Use tag objects, write UI functions **Our focus today**
- Step 4. Author HTML templates
- Step 5. Create custom inputs/outputs, wrap existing CSS/JS libraries and frameworks

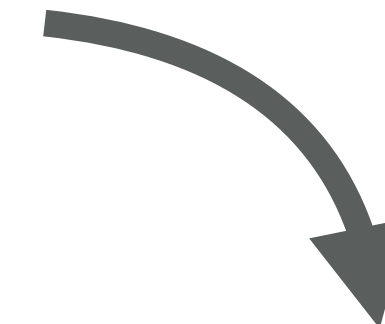
High level
view

MULTIPLE LEVELS OF ABSTRACTION

High-level funcs
`fluidRow(...)`



htmltools tags
`div(class="row", ...)`



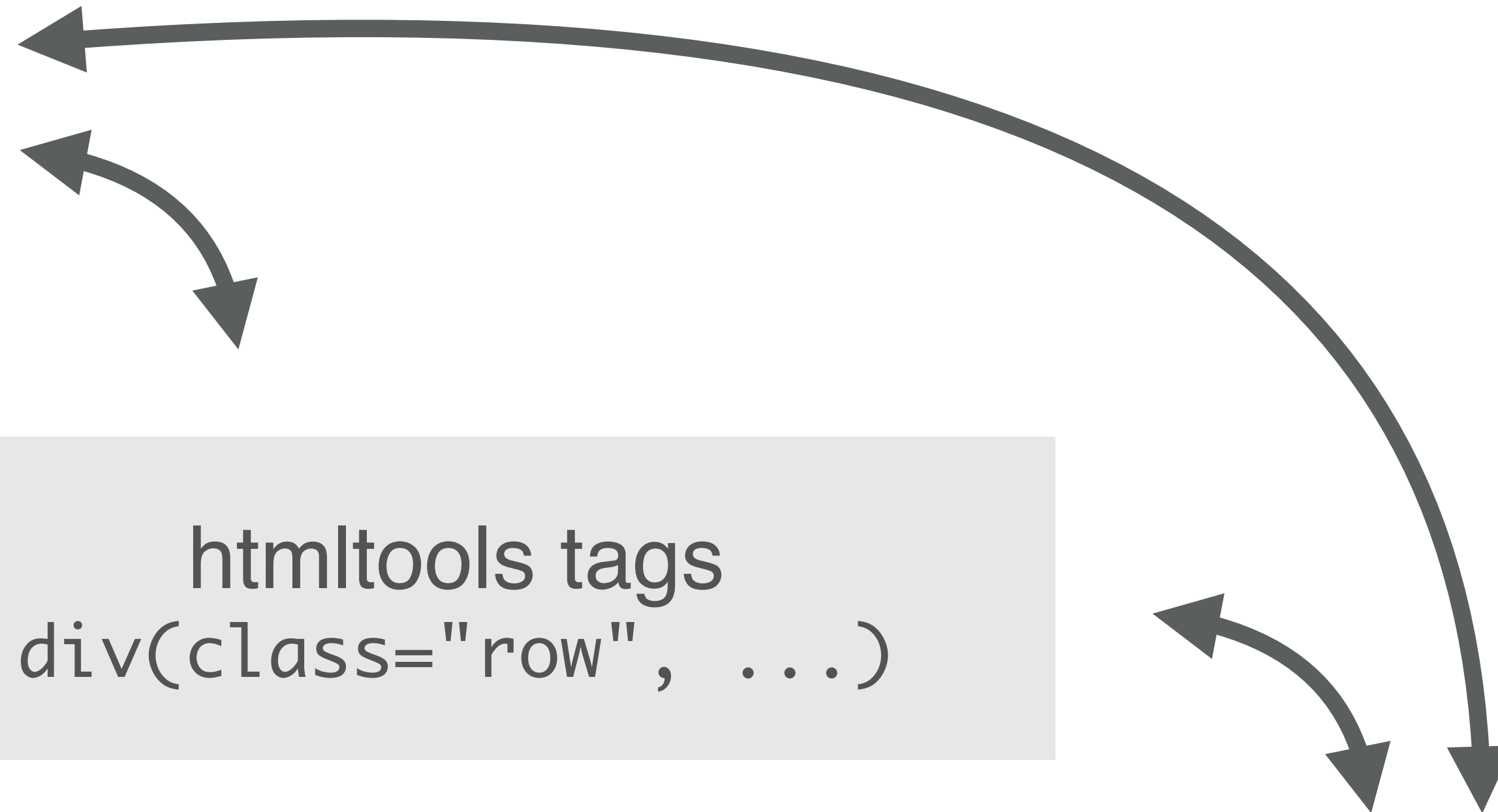
Raw HTML markup
`<div class="row">...</div>`

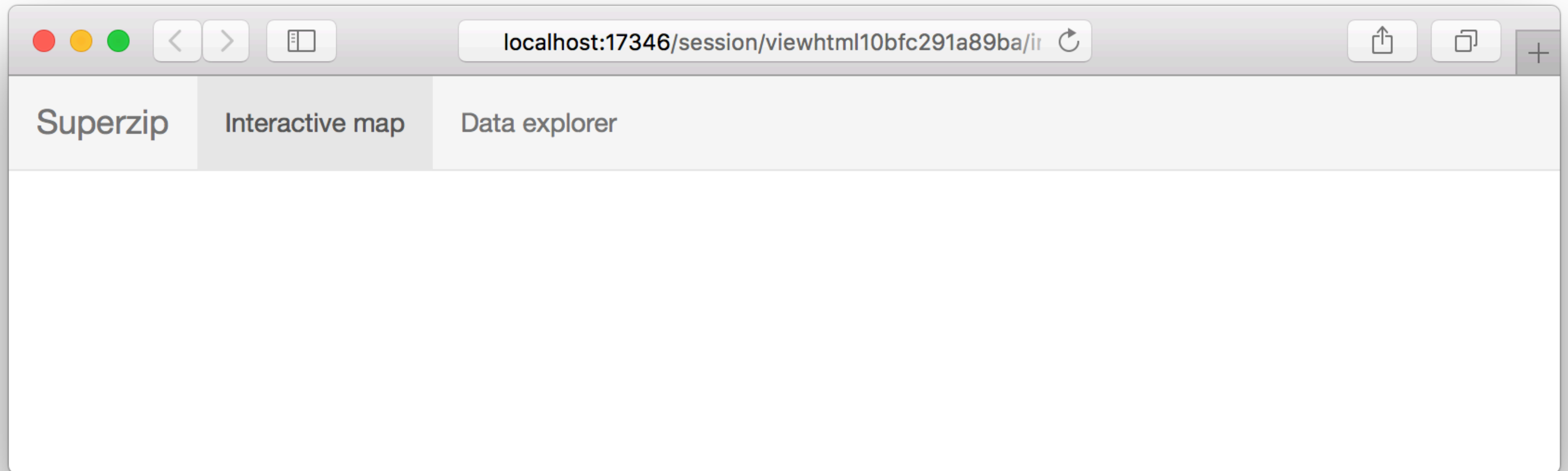
MIX AND MATCH FREELY

High-level funcs
`fluidRow(...)`

htmltools tags
`div(class="row", ...)`

Raw HTML markup
`<div class="row">...</div>`





RAW HTML

- Pros
 - Can do anything that's possible in a web page
 - Comfortable for designers, web developers
- Cons
 - Unfamiliar for many R users
 - Potentially lots of HTML needed for conceptually simple tasks
 - CSS/JavaScript dependencies must be handled manually

```
<nav class="navbar navbar-default navbar-static-top" role="navigation">
  <div class="container">
    <div class="navbar-header">
      <span class="navbar-brand">Superzip</span>
    </div>
    <ul class="nav navbar-nav shiny-tab-input" id="nav">
      <li class="active">
        <a href="#tab-5158-1" data-toggle="tab" data-value="1">Interactive</a>
      </li>
      <li>
        <a href="#tab-5158-2" data-toggle="tab" data-value="2">Map</a>
      </li>
      <li>
        <a href="#tab-5158-3" data-toggle="tab" data-value="3">About</a>
      </li>
    </ul>
  </div>
</nav>
<div class="container-fluid">
  <div class="tab-content">
    <div class="tab-pane active" data-value="Interactive">
      <div class="outer">
        <div id="map" style="width:100%; height:100%; border: 1px solid black;">
          <div class="panel panel-default draggable" id="map">
            <img alt="Map of the world" data-bbox="100 100 900 900"/>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

HTMLTOOLS OBJECTS

- HTML-generating R functions
- Pros
 - All the power of HTML, but looks like R
 - Automated CSS/JS dependency handling
 - More composable, programmable than HTML
- Cons
 - Easy to misplace commas
 - Almost as verbose as raw HTML

```
nav(class="navbar navbar-default navbar-static-top", rol
  div(class="container",
    div(class="navbar-header",
      span(class="navbar-brand", "Superzip")
    ),
    ul(class="nav navbar-nav shiny-tab-input", id="nav",
      li(class="active",
        a(href="#tab-5158-1", `data-toggle`="tab", `data
      ),
      li(
        a(href="#tab-5158-2", `data-toggle`="tab", `data
      ),
      li(
        a(href="#tab-5158-3", `data-toggle`="tab")
      )
    )
  )
)
```

HIGH LEVEL FUNCTIONS

- Functions that return htmltools objects
- Pros
 - Less code, clearer intent
 - Anyone can make their own
- Cons
 - Still have to watch out for commas
 - Less flexible

```
navbarPage("Superzip", id = "nav",  
  tabPanel("Interactive map", ...),  
  tabPanel("Data explorer", ...)  
)
```

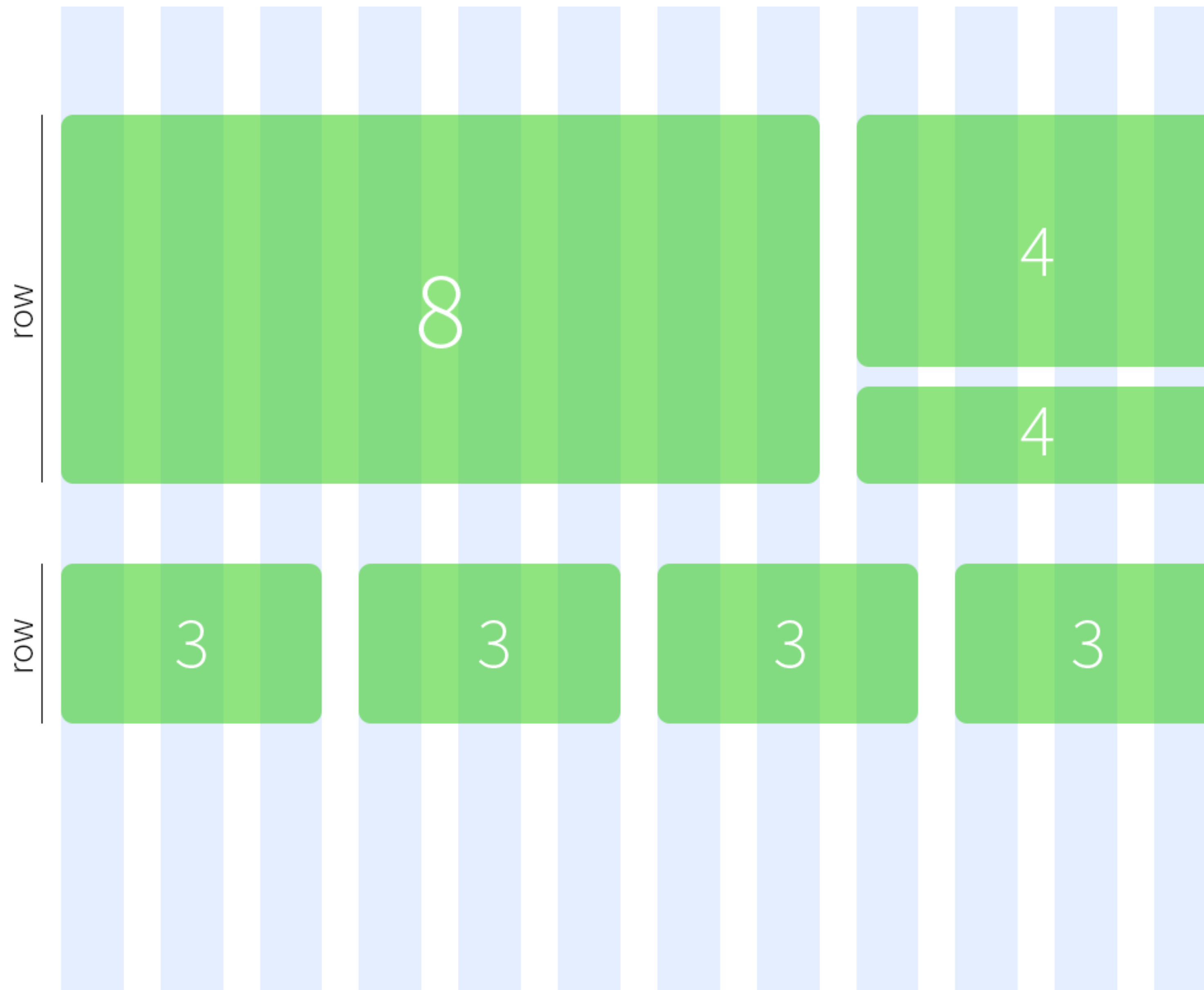
Using Shiny built-ins

SHINY UI BUILT-INS

- Bootstrap grid framework – `fluidPage`, `fixedPage`, `fluidRow`, `column`
- Containers – `wellPanel`, `absolutePanel`, `fixedPanel`
- Navigation panels – `tabsetPanel`, `navlistPanel`, `navbarPage`
- Fill layouts (Shiny 0.13+) – `fillPage`, `fillRow`, `fillCol`
- Modals and notifications (Shiny 0.14+) – `showModal`, `modalDialog`

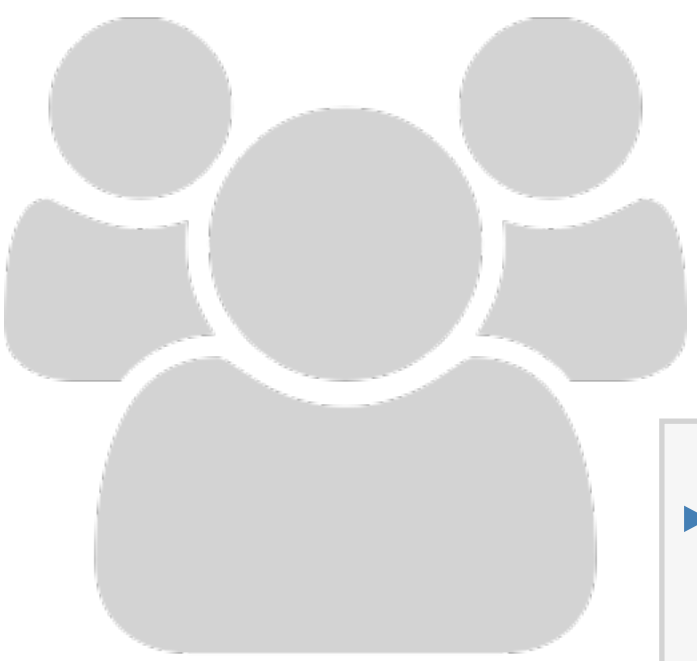
BOOTSTRAP GRID FRAMEWORK

- Every page has 12 invisible columns
- Each column of content must span an integral number of columns
- Simple R API for implementing Bootstrap grid
 - `fluidPage(...)` wraps the entire page
 - `fluidRow(...)` wraps each row's column
 - `column(width, ...)` wraps each column's content



```
ui <- fluidPage(  
  fluidRow(  
    column(8, item1),  
    column(4, item2, item3),  
  ),  
  fluidRow(  
    column(3, item4),  
    column(3, item5),  
    column(3, item6),  
    column(3, item7)  
  )  
)
```

EXERCISE



- ▶ Modify `ui_01.R` to display the two outputs next to each other (instead of above and below)
- ▶ Assign the left output to be 5 columns wide, and the right output to be 7 columns wide
- ▶ See what happens as you change the width of the browser window

3_m 00_s



SOLUTION

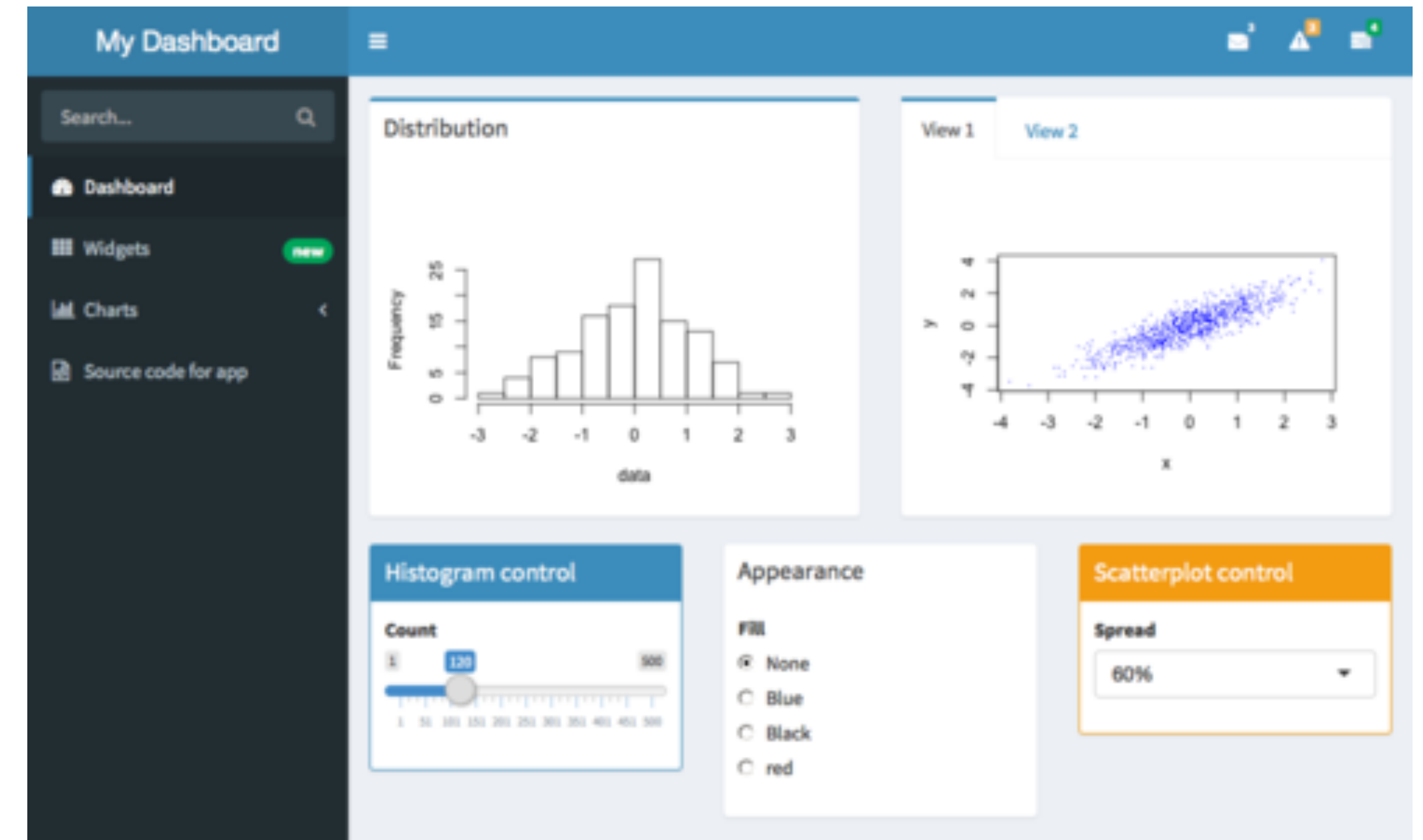
Solution to the previous exercise

`ui_02.R`

Using external
packages

EXTERNAL PACKAGES

- shinydashboard



EXTERNAL PACKAGES

- shinydashboard
- shinythemes

The image displays a collage of Shiny dashboard examples, illustrating the use of external packages like shinydashboard and shinythemes. The dashboards are layered, showing different themes and layouts.

Themes shown:

- Darkly:** A dark-themed dashboard with a sidebar on the left containing file, text, and slider inputs, and a main content area with buttons.
- United:** A light-themed dashboard with a top navigation bar and a sidebar on the left.
- Flatly:** A light-themed dashboard with a top navigation bar and a main content area.

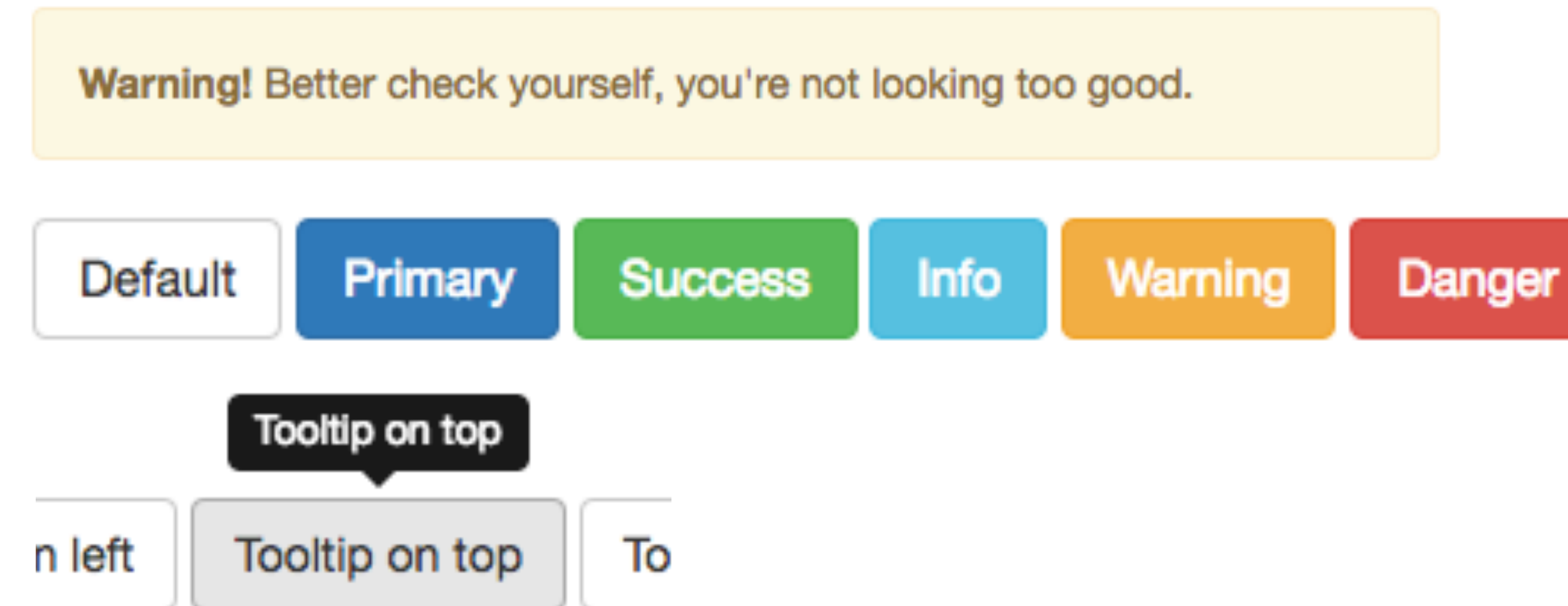
Components and Data:

- File input:** A "Browse..." button and a "No file selected" message.
- Text input:** A text box containing the word "general".
- Slider input:** A slider ranging from 1 to 100, with the value 30 selected.
- Default action button:** A button labeled "Search".
- ActionButton with CSS class:** A button labeled "Action button".
- Table:** A table with two columns, "speed" and "dist", showing data for different categories.
- Verbatim text output:** A text box displaying the output "general, 30, NULL".
- Headers:** A list of headers labeled "Header 1" through "Header 5".

| speed | dist |
|-------|-------|
| 4.00 | 2.00 |
| 4.00 | 10.00 |
| 7.00 | 4.00 |
| 7.00 | 22.00 |

EXTERNAL PACKAGES

- shinydashboard
- shinythemes
- shinyBS (@ebailey78)



EXTERNAL PACKAGES

- shinydashboard
- shinythemes
- shinyBS (@ebailey78)
- shinytoastr (@gaborcsardi)



Cannot access database



Clock mismatch detected



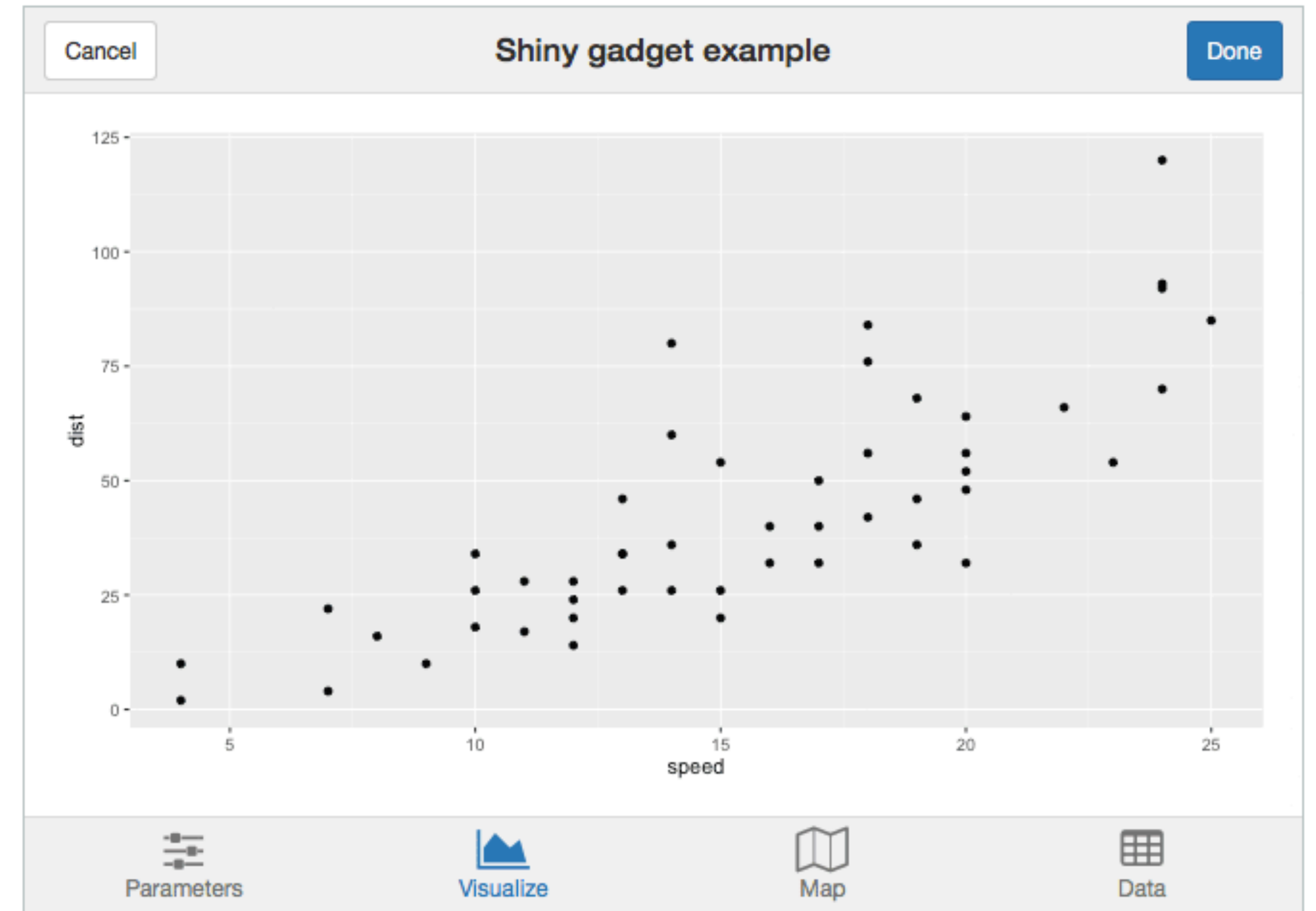
Using the test database



Database updated

EXTERNAL PACKAGES

- ▶ shinydashboard
- ▶ shinythemes
- ▶ shinyBS (@ebailey78)
- ▶ shinytoastr (@gaborcsardi)
- ▶ miniUI (for mobile devices or Shiny Gadgets)



EXTERNAL PACKAGES

- shinyjs (@daattali)
- Perform many UI-related JavaScript operations from R

| Function | Description |
|---|--|
| <code>show / hide / toggle</code> | Display or hide an element (optionally with an animation). |
| <code>hidden</code> | Initialize a Shiny tag as invisible (can be shown later with a call to <code>show</code>). |
| <code>enable / disable / toggleState</code> | Enable or disable an input element, such as a button or a text input. |
| <code>disabled</code> | Initialize a Shiny input as disabled. |
| <code>reset</code> | Reset a Shiny input widget back to its original value. |
| <code>delay</code> | Execute R code (including any <code>shinyjs</code> functions) after a specified amount of time. |
| <code>alert</code> | Show a message to the |
| <code>html</code> | Change the text/HTML of an element. |
| <code>onclick</code> | Run R code when a specific element is clicked. Was originally developed with the sole purpose of running a <code>shinyjs</code> function when an element is clicked, though any R code can be used. |
| <code>onevent</code> | Similar to <code>onclick</code> , but can be used with many other events instead of click (for example, listen for a key press, mouse hover, etc). |
| <code>addClass / removeClass / toggleClass</code> | add or remove a CSS class from an element. |
| <code>runjs</code> | Run arbitrary JavaScript code. |
| <code>extendShinyjs</code> | Allows you to write your own JavaScript functions and use <code>shinyjs</code> to call them as if they were regular R code. More information is available in the section "Calling your own JavaScript functions from R" below. |

EXERCISE



- ▶ Modify `movies_12.R` to use a Bootstrap theme
 - ▶ Use the "Live theme selector" feature in `shinythemes` in your own app
 - ▶ Once you've decided on a theme, remove the theme selector and apply your chosen theme permanently
- ▶ See shinythemes instructions at:
<https://rstudio.github.io/shinythemes/>

5_m 00_s



SOLUTION

Solution to the previous exercise

`movies_13.R`

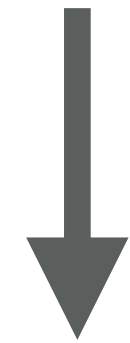
Using `htmltools`
tag objects

AN API FOR COMPOSING HTML

- ▶ When Shiny was born, it came with a sub-package for composing HTML
- ▶ These functions were so useful, we extracted them out into a separate package: `htmltools`
- ▶ Now used by R Markdown and `htmlwidgets` as well

HTML BASICS

```
<a href="https://www.rstudio.com">RStudio</a>
```



[RStudio](https://www.rstudio.com)

HTML BASICS

```
<a href="https://www.rstudio.com">RStudio</a>
```

End tag

Start tag

Child content

Attribute name

ANATOMY OF A TAG

```
<a href="https://www.rstudio.com">RStudio</a>
```

Tag name

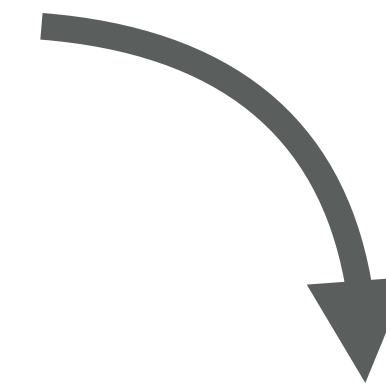
Attribute value

Creates an anchor whose
hyperlink reference is the URL
<https://www.rstudio.com>

ANATOMY OF A TAG

- Text can contain tags
- Tags can optionally contain text and/or other tags
- Each start tag can have zero or more attributes

```
<div class="panel panel-default">  
  <div class="panel-heading">  
    <h3 class="panel-title">Panel title</h3>  
  </div>  
  <div class="panel-body">  
    Panel content  
  </div>  
</div>
```



Panel title

Panel content

LOOKS LIKE R, MEANS HTML

```
<div class="panel panel-default">  
  <div class="panel-heading">  
    <h3 class="panel-title">  
      Panel title  
    </h3>  
  </div>  
  <div class="panel-body">  
    Panel content  
  </div>  
</div>
```

```
div(class="panel panel-default",  
  div(class="panel-heading",  
    h3(class="panel-title",  
      "Panel title",  
    )  
  ),  
  div(class="panel-body",  
    "Panel content"  
  )  
)
```

USING TAG FUNCTIONS

- ▶ Many common tags are exported as functions by `htmltools` and `shiny` (`p`, `h1-h6`, `a`, `br`, `div`, `span`, `img`)
- ▶ All other tags can be accessed via the `tags` object. E.g., `Item 1` → `tags$li("Item 1")`
- ▶ If you have lots of HTML to write, you can use the `withTags` function—it makes the `tags$` prefix optional.

```
withTags(  
  ul(  
    li("Item 1"), li("Item 2")  
  )  
)
```

USING TAG FUNCTIONS

- ▶ All tag functions behave the same way
 - ▶ Call the function to create a tag object
 - ▶ Named arguments become attributes
 - ▶ Unnamed arguments become children

TAG ATTRIBUTES

- Any valid HTML attribute name can be used (use quotes if the name has dashes, e.g. "data-toggle"="dropdown")
- Valid tag attribute values are:
 - NULL (omit the attribute)
 - NA (the attribute should be included with no value)
 - Single-element character vector (or something to be coerced to character)

```
tags$input(type = "checkbox",  
  disabled = if (disabled) NA # else NULL  
)
```

TAG CHILDREN

- Valid tag children are:
 - Tag objects
 - Single-element character vectors (treated as text)
 - NULL (silently ignored)
 - Raw HTML (see `?htmltools::HTML`)
 - Lists of valid tag children (recursive!)

USING TAGS

- Tags are made using normal R functions that take normal parameters and return normal values! You can do R-like things to them:
`tags$ul(lapply(1:10, tags$li))`
- Print tag objects at the console to see their HTML source
 - Call `print(x, browse = TRUE)` to see their rendered view instead
 - Use `htmltools::browsable()` to make an object show its rendered view when printed, by default
 - If your top-level object is a list, you'll need to wrap in `tagList(...)` to get the right behavior at the console (or in an R Markdown doc)

EXERCISE



- ▶ Open `ui_03.R`.
- ▶ Replace `includeHTML("youtube_thumbnail.html")` with the equivalent `htmltools` tag objects.
 - ▶ Hint: Take a look inside `youtube_thumbnail.html`.
- ▶ If you get that working, take the next step and define an R function that takes a YouTube URL, a title, and a description, and returns a thumbnail frame like the one you created.

5_m 00_s



SOLUTION

Solutions to the previous exercise

`ui_04.R`

`ui_05.R`



DEMO

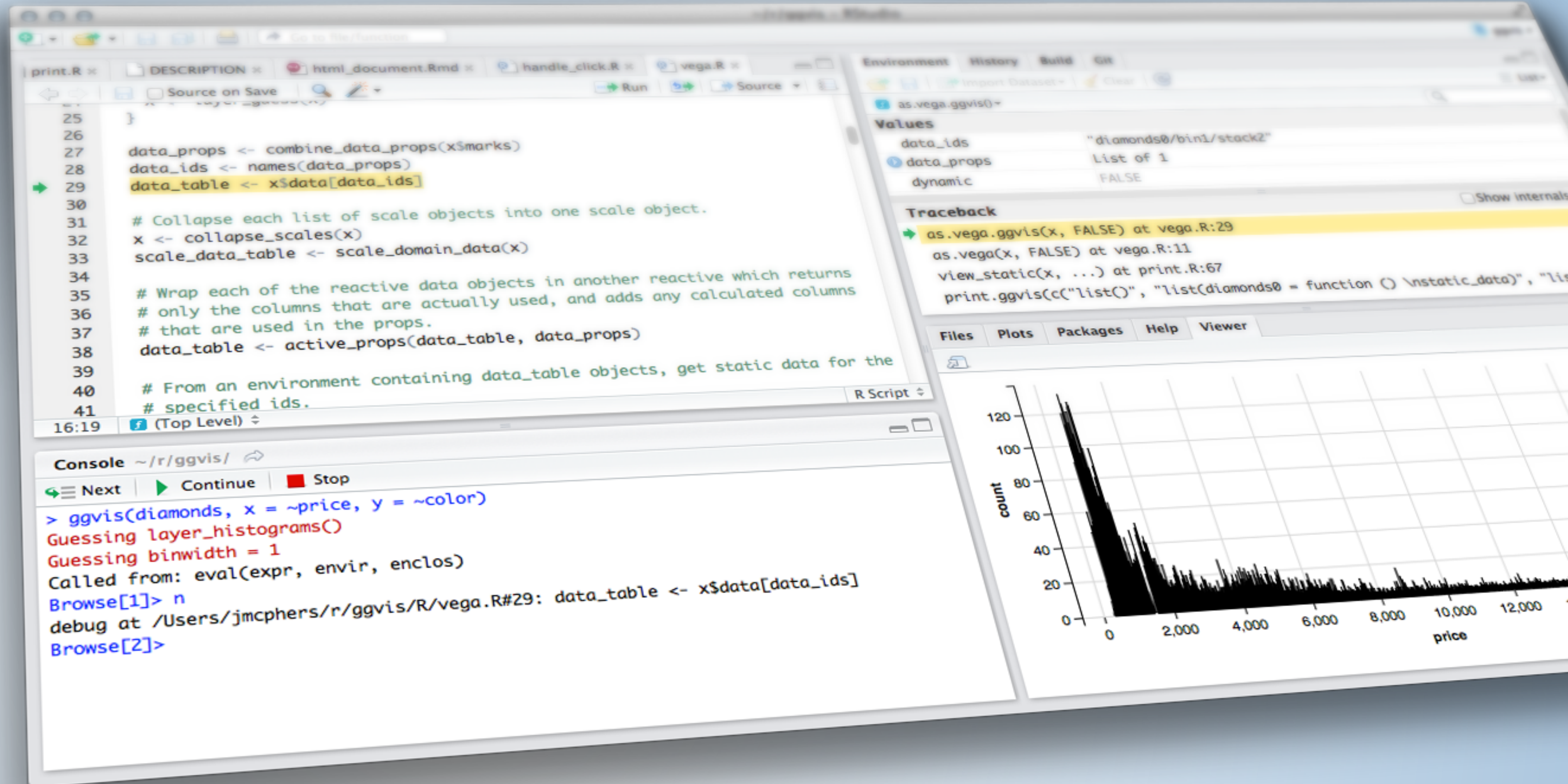
Layouts

Using


raw HTML

USING RAW HTML

- Incorporate tiny amounts of HTML using inline string literals wrapped in HTML()
 - `div(HTML("This is HTML"))`
- For chunks of (static) HTML, use `includeHTML` (or similar `includeCSS`, `includeScript`)
 - `div(includeHTML("file.html"))`
- Or go the other way, with the HTML Templates feature: start with HTML, and embed R expressions that yield tag objects



UNDERSTANDING UI

- 
- ▶ Turning in your homework
 - ▶ Upload your app to shinyapps.io
 - ▶ Set the url to the repository in your GitHub repo:
 - ▶ Edit button on the top right near the Settings tab
 - ▶ Email me and Malvika
 - ▶ gla@andrew.cmu.edu, malvikas.andrew.cmu.edu
 - ▶ Include your GitHub repo link and the shinyapps.io link